Web Applications A.Y. 2023-2024

Homework 1 – Server-side Design and Development

Master Degree in Computer Engineering
Master Degree in Cybersecurity
Master Degree in ICT for Internet and Multimedia

Deadline: 29 April, 2024

Group Acronym	RanaMelone	
Last Name	**First Name**	**Badge Number**
Torres-Pardo López	Isabel	2099920
Alcázar Pérez	Jesús	2099107
Jassal	Arjun	2095488
Valentinuzzi	Andrea	2090451
Brejc	Giovanni	2096046
Martín Bacas	Álvaro	2099592

1 Objectives

Lyrify is designed to simplify accessing song lyrics. With its clean interface and straightforward search function, users can quickly find lyrics for their favorite songs. The platform offers basic personalization options like saving favorite songs and creating playlists. It aims to be a user-friendly platform for music lovers to enjoy and manage song lyrics hassle-free.

2 Main Functionalities

Lyrify+ has two main users: users can sign up as an artist or as a normal user.

In order to access Lyrify+, artists and users need to sign up providing their username for the app, an email and a password. In addition to this, artists need to select the artist check field before creating the account and later on they will have to ask for a verification in order to be able to upload songs or albums.

As an artist, once verified, they are able to upload their own songs, providing the name, release date, genre, link to the lyrics and the link to the youtube video; create albums, providing also the name for it, the songs and the data required for them, a cover and a release date; and a small description talking about the type of music the do or biography. Albums once created cannot be changed. Artist are only able to upload data, not to consume it, for that they will need to create an user account.

As an user, it is possible to choose the avatar you like for your profile, create playlists with your favorite songs, being able to add/remove songs whenever you want, add Albums to your library, look for suggestion on the main page and search for songs by name, genre or artists.

3 Data Logic Layer

3.1 Entity-Relationship Schema

The ER schema contains the following entities:

- **Users**: describes the user that will use our web application. Each user has as a primary key a serial ID that is given the moment the account is created. For each user we also record the email, their password, which can have length between 8 and 24 characters, and their username and an avatar image provided from a list of possible options.
- **Songs**: each song is uniquely determined by a serial ID number, given the moment the song is uploaded. It also contains some data of the song, such as the tittle, YouTube link to the music video, release date, the duration of the song, list of genres the song belongs to and the lyrics of the song.
- **Playlist**: is a list of songs that a user creates. As a primary key it has an ID and as attribute a tittle. Also, as a foreign key we have the User ID, which refers to the owner and creator of the playlist.
- **Artist**: similar to an user but need a verification in order to create the account. As identifier it has a serial ID number and has a name, an image, a biography, a verification boolean in order to upload songs and a language in which most of its songs are written. Also an artist has an email and a password, which can have a length between 8 and 24 characters
- **Album**: is a collection of songs that are created by an artist. They are recognized by a serial ID and also have a tittle, a image that is the cover of the album, a release date and a list of genres it belongs to.

Now we dive into the relationships between our entities:

- **Creates**: an user can create a playlist, which is only owned and created by one user, this is that there are not collaborative playlist. An user can have none playlist or as much as he wishes. In summary, is a 1-N relationship.

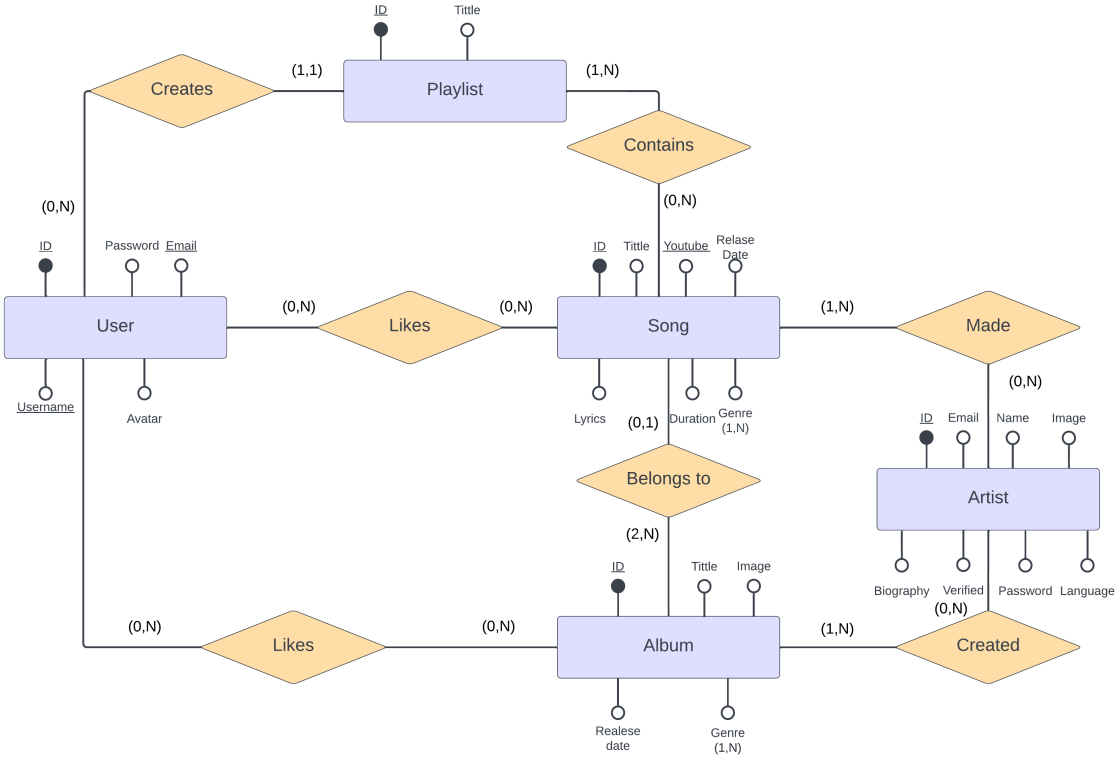

Figure 1: ER Schema diagram

- **Contains**: is the relation between playlist and song, in which is described that a playlist must have at least one song and no maximum amount; and a song can be or not be in a playlist. It is a N-N relationship. In the SQL database is represented by the name "SongsInPlaylists".
- **Likes**: an user has the ability to like songs. A new user will begin without any liked songs and as the user likes more songs, the list will grow. A song can either be liked or not by an user. In the SQL database is represented by the name "LikedSongs".
- **Likes**: an album can be liked by many different users, and a user can like or not any album, new users will not like any album until they like the first one. In the SQL database is represented by the name "LikedAlbums".
- **Belongs to**: A song can belong to one album or be a single, which means that it does not belong to any album. Albums must at least have two songs, in case of one song only, it would be consider to be a single song and not an album itself. In the SQL database is represented by the name "SongsInAlbums".
- **Made**: is the relationship between songs and artists. An artist can make no songs (can not upload songs until the verification is completed) or have plenty of them. A song must be created for at least one artist, and in case of a collaboration, more than one artist.
- **Created**: Albums are created by at least one artist, and must be uploaded by the artist itself. An artist can have no albums (just upload single songs) at all or to have one or more albums.

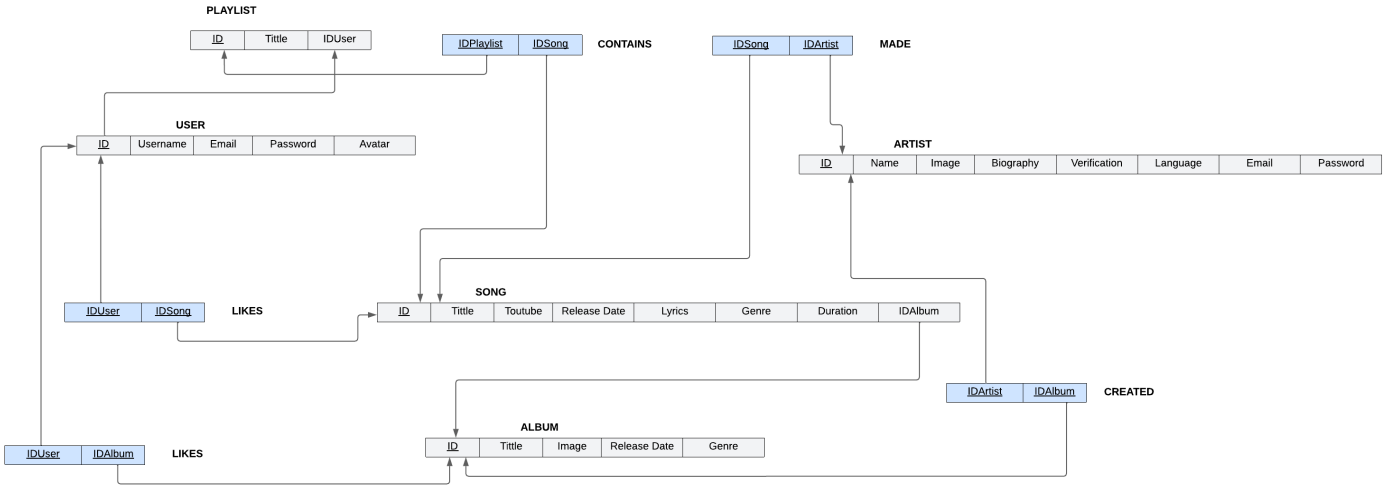

Figure 2: ER Tables Diagram

3.2 Other Information

It is possible to interact with most of the entities (insertion, deletion and update) directly via interface. The only exception is for the album entity, which can only be created and deleted by an artist.

There is a custom enumeration called Genre, in which we save a list of different genres of music, as example 'Pop' and 'Rock'. In the enumeration, the value 'Other' is established by default, so if during the creation of a song or an album the artist doesn't specify a genre, the type 'Other' will be written.

4 Presentation Logic Layer

Lyrify+ has two main areas: the user area and the artist area. Both are accessible through the login page where the user can decide to login as an user or as an artist, according to their role. The accessible pages for users can only be seen by users, idem for the artist, the only common page for both type of users is the login page. Now we define the pages to be developed in Lyrify+.

- Login / Register page: first shown page to all users. They will be able to choose to login as an user/artist or to register as a new user/artist.
- Homepage: main page showed when a user's is logged in.
- User profile page: page that shows the user personal data and account options.
- Artist profile page: page that shows the artist's personal data, upload options and account options.
- Artist page: page that shows the public data of an artist, including biography, songs and albums.
- Upload song page: page where it's possible to upload songs, only accessible by verified artists.
- Upload album page: page were it's possible to upload albums, only accessible by verified artists.
- Library page: page showing users' playlists, liked songs and added albums.
- Song page: page that shows information about the song.
- Playlist/Album page: page that shows information about the playlist or the album.

4.1 Login / Register Page (Interface Mockup)

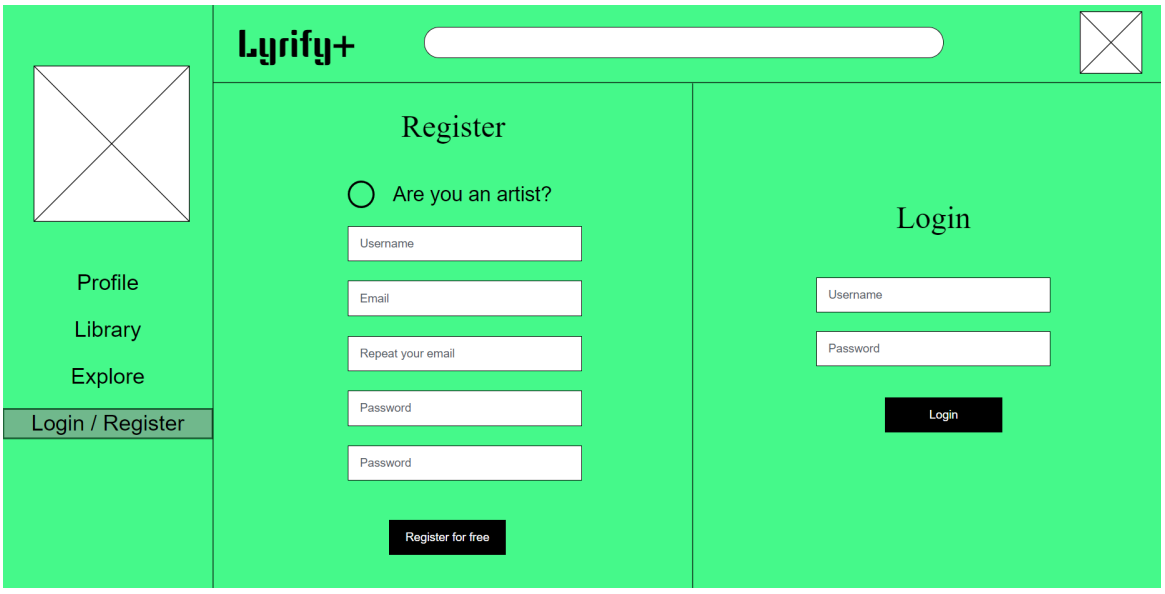

Figure 3: Login page

The Login/Register page is the first page that appears when accessing Lyrify+ on the browser. Here, the user can either register as a new user or artist or login if they already have an account. For the register form, the user will be asked to: mark the radial button in case he wants to create an account as an artist, enter an username, an

email and confirm it and a password also with its confirmation. The username to be provided can have as much as 50 characters, the email is required to have the typical email schema and the password will be required to be between 8 to 25 characters. By clicking the 'Register for free' button the user's credentials will be added to our database and the user will be redirected to the homepage, or to the artist profile page in case they registered as an artist. For the login form the user is required to enter their credentials (username and password) and then click the 'Login' button that will check if the credentials are correct and if they are in our database. Once completed the users will be redirected to the homepage and the artists will be redirected to the artist profile page.

4.2 Homepage (Interface Mockup)

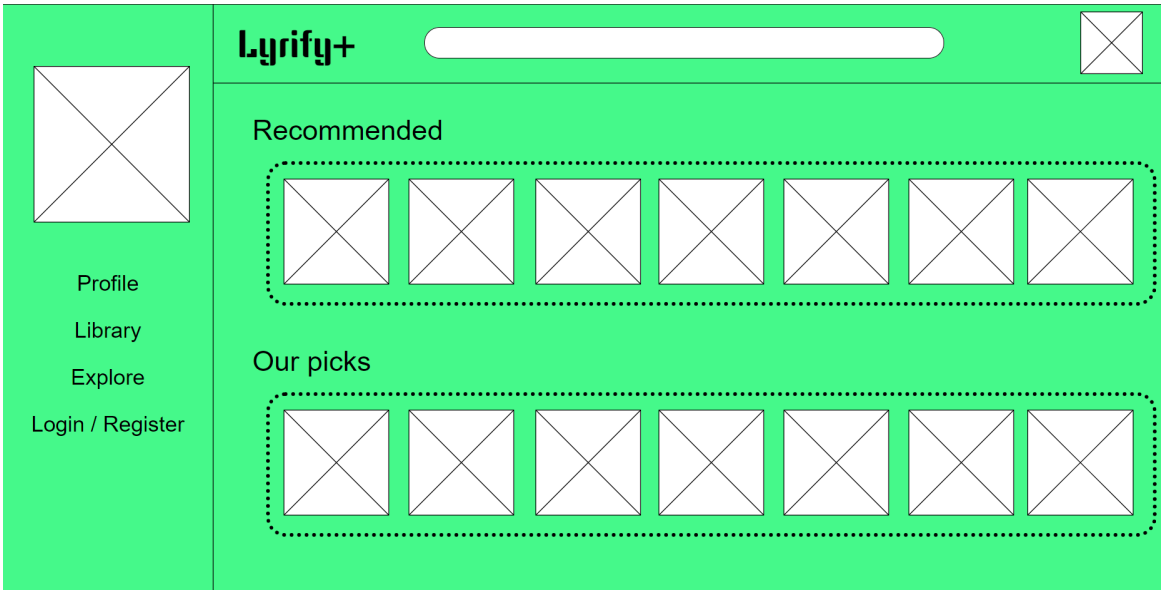

Figure 4: Homepage

On the Homepage, the screen will show a menu on the left side that includes the app's logo, a button for accessing the user's profile, the library page, an explore page for discovering new songs and a logout button in case they want to leave the application. On the top row they will be able to see the app's name, a search bar for searching songs in the database and their profile photo on the top right corner to show that they are logged in. Then, in the center of the page, the user will see two main rows showing our recommendations, picked by the algorithm: as of now the algorithm reads from the database songs and albums not older than 1 year, but this could be changed in the future.

4.3 User Profile Page (Interface Mockup)

On the user's profile page, the menu on the left side will be repeated but with the 'Profile' button darkened to indicate that the user is in that page. Then, the top row is also shown including the app's name, search bar and profile photo. The center of the page will be divided in two parts, the user's information part and the account settings part. For the user's information part, the profile photo will be displayed in a bigger size along with the username and the mail. For the account settings part, there will be an option for changing the password by filling both fields and then clicking on the 'Change password' button, that will update our database with the new credentials for the user. In addition, in case the user no longer wants to keep using our app, there will be a 'Delete account' button that will delete all the user's data from our database and will redirect the user to the Login / Register page.

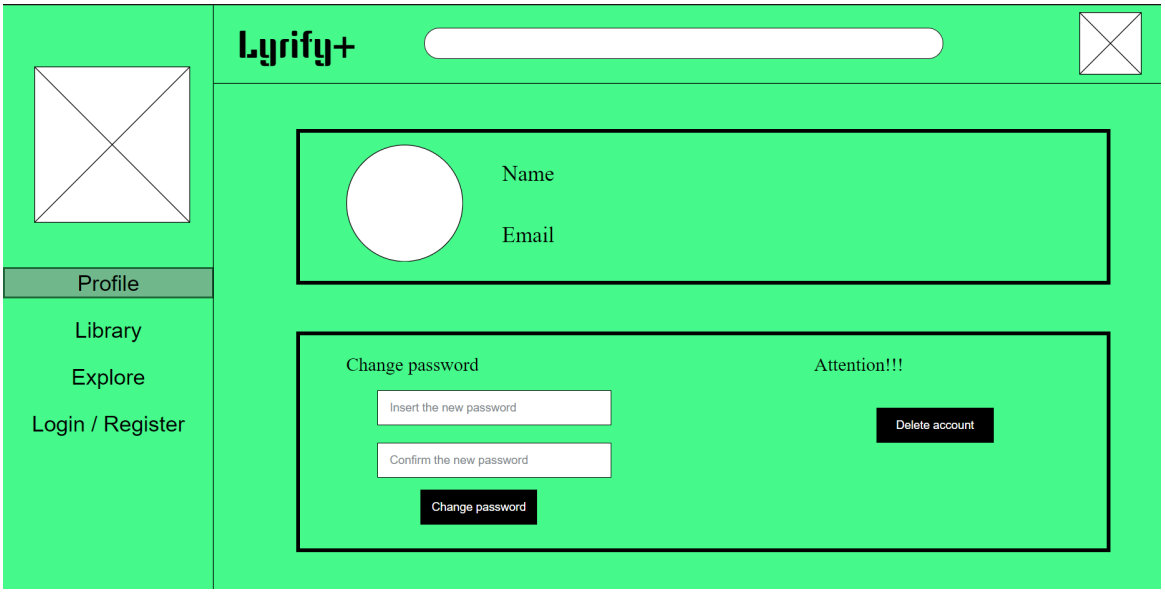

Figure 5: User profile page

4.4 Artist Profile Page (Interface Mockup)

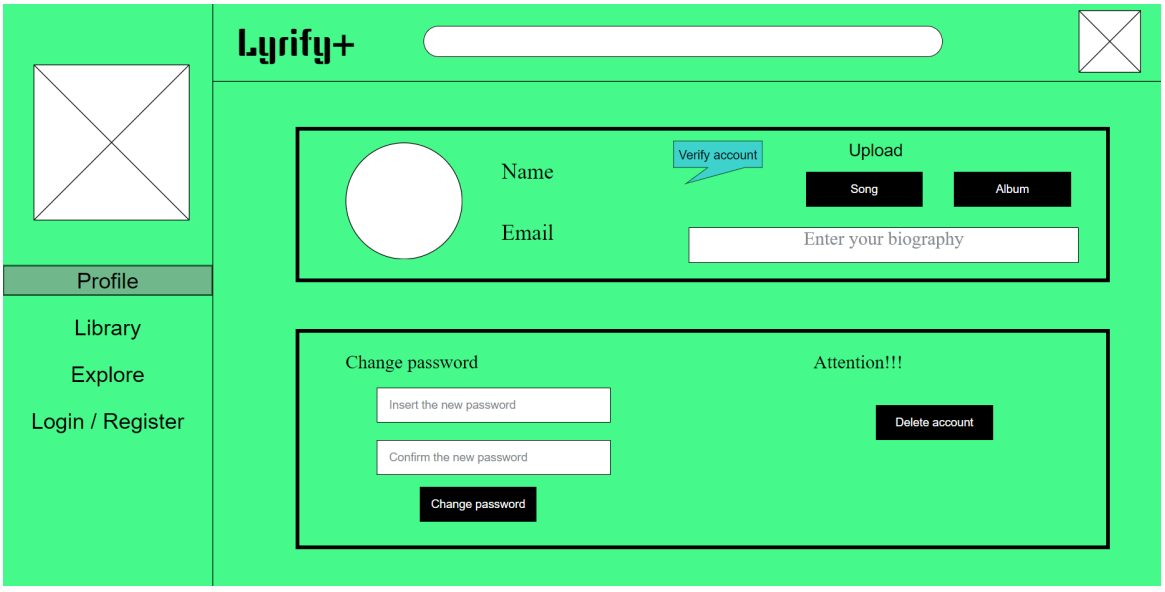

Figure 6: Artist profile page

On the artist's profile page, the menu on the left side and the top row will remain the same as in the user's profile page. The center of the page will be divided in two parts, the artist's information and options part and the account settings part.

For the artist's information part, the profile photo will be displayed in a bigger size along with the username and the mail. In addition, it includes the verify button, that will disappear once the account is verified; and the buttons for uploading either songs or albums and the biography area for the artist to describe himself.

For the account settings part, we maintained the same fields as in the user's profile page, an option for updating the password and for deleting the account.

4.5 Artist Page (Interface Mockup)

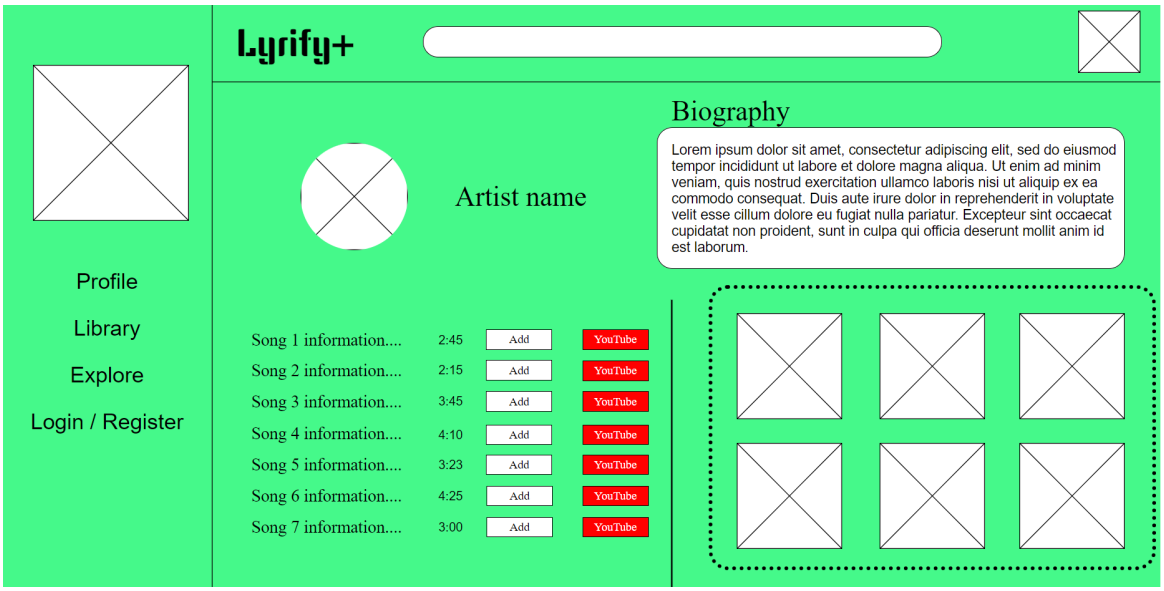

Figure 7: Artist page

For the artist's page, which is not the same as the artist's profile page, it will be shown the public information from the artist. We maintain the design for the side panel and the top row as for the other pages. Then, in the center of the page the following information is shown:

1. Top left corner: Artist's name and profile photo
2. Top right corner: The artist's biography
3. Down left corner: A list with the artist's songs showing the name of each of them, their duration, a button for adding them to a playlist and a YouTube button to redirect the user to the song's YouTube video. In addition, by clicking on the name of the song, the user will be redirected the song's information page, which will include the lyrics of it.
4. Down right corner: A grid with the artist's released albums showing only the album's cover, that by clicking on them will redirect the user to the album's information page, with their respective information and songs belonging to the album.

4.6 Upload Song Page (Interface Mockup)

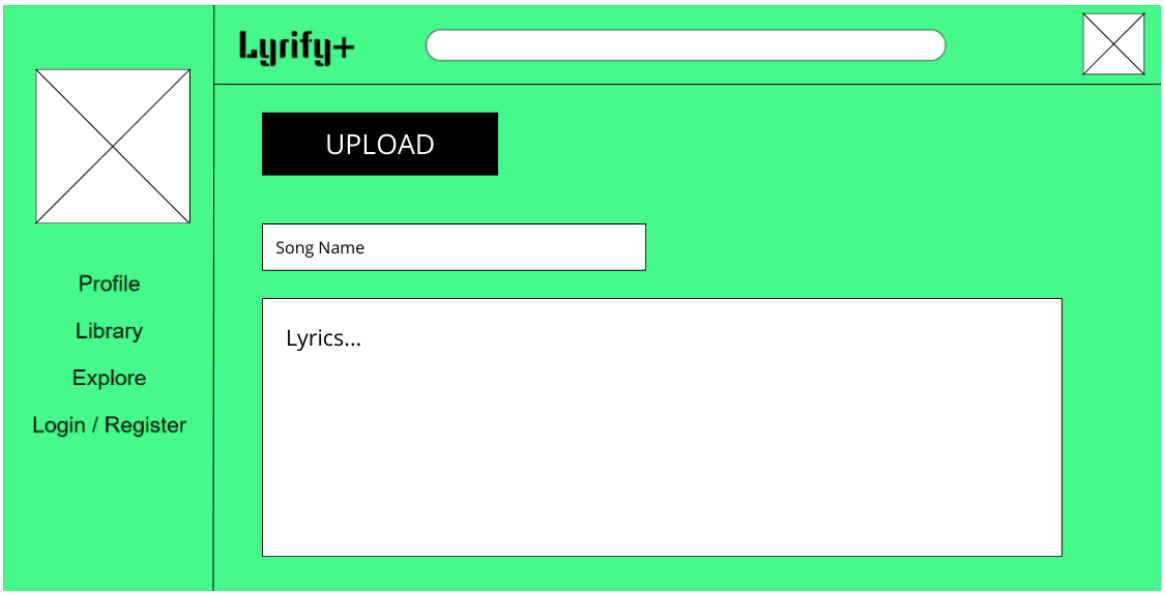

Figure 8: Upload song page

This page is simpler as it has the typical design of the others for the side and the top row. In the center of the page, we can find the upload button, for adding the song to our database, and two fields for entering the song's name and lyrics.

4.7 Upload Album Page (Interface Mockup)

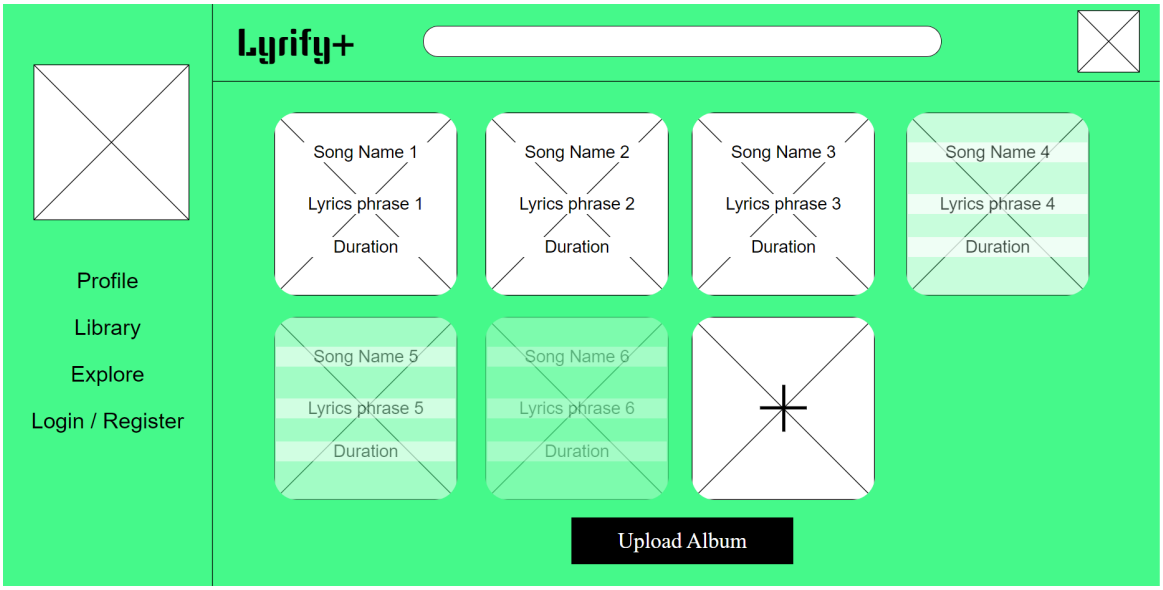

Figure 9: Upload album page

This page is simpler too as it has the typical design of the others for the side and the top row. In the center of the page, we can find some squares that represents the songs added to the album. The page will begin only with the square that has a plus in the center. By clicking on it, the page will redirect you to the upload songs page in order to enter the rest of the song's information. When done, the page will redirect you back to the album, and one square with the basic info of the song the artist just entered will be displayed, and next to it the add song square. This way the page will be filling up with the songs the artist enters. Once the album is completed, the artist will need to click on the 'Upload album' button to save it in our database and will be redirected to the artist's profile page.

4.8 Library Page (Interface Mockup)

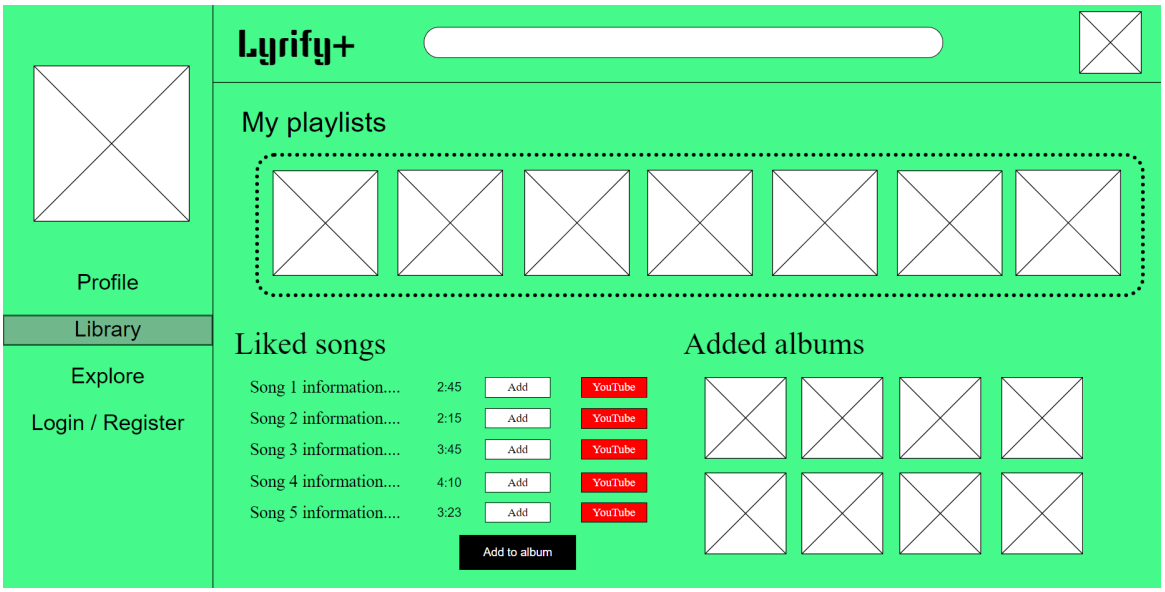

Figure 10: Library page

For this page, we still maintain the design for the side and top panels, with the exception on the darkened 'Library' part to show that the user is on the library page. In the center of the page, there will be displayed a row with the user's playlist that will be empty in the beginning and will grow as the user makes them. On the bottom part, the user will be able to see a small list of the liked songs, showing the names, duration, possibility to add to a playlist and YouTube video button; on the other side a list of the added albums will be displayed too, showing only the albums' covers.

4.9 Song Page (Interface Mockup)

For the song page, there will still be the top and side panels as in the other pages. In the center, the user will be able to see the name of the son, next to it a button to like the song, the YouTube video button, author's name and a button for adding the song to a playlist. Below this, the lyrics of the song will be displayed, and on the bottom right corner the duration of the song will be shown.

4.10 Playlist / Album Page (Interface Mockup)

For this page, the general design is maintained as always, regarding the side and top panels. Then, the info displayed in the page will depend on from where it is opened, because it will display info for playlists and albums.

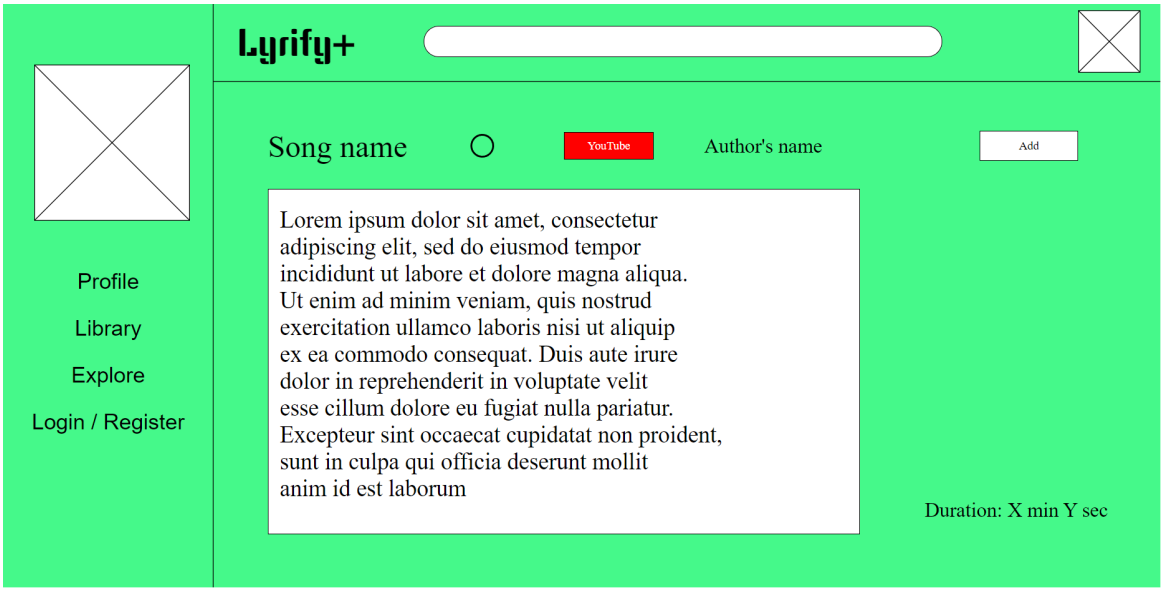

Figure 11: Song page

Figure 12: Playlist / Album page

As an album page, the album's name will be displayed on top, next to it, an added button will appear selected in the case the album is added by the user or empty in case the user has not selected yet the album. On the right side, the album's cover photo, name, author and complete duration will be displayed. And, in the center of the page, the list of songs belonging to the album will be displayed, showing the songs' names, durations, add buttons and YouTube video buttons.

As a playlist page, the page will show the playlist's name, and the list of songs with all the respective information regarding the songs.

5 Business Logic Layer

5.1 Class Diagram

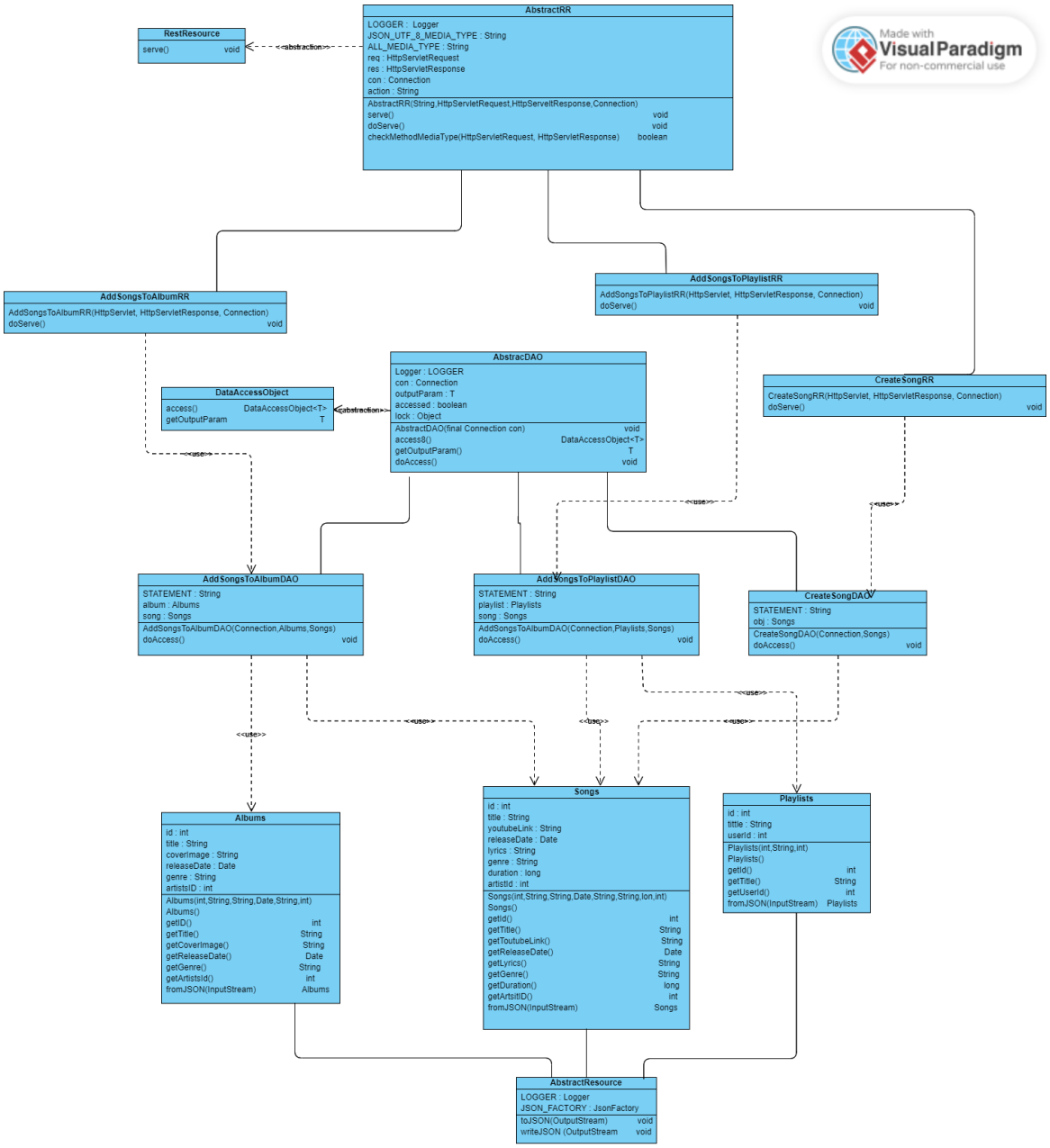

Figure 13: Class Diagram

The class diagram contains (some of) the classes used to handle three types of resources: songs, albums and playlists. It is possible to observe that we have REST classes to handle all of the actions we are showing in the diagram. Each REST extends the AbstractRR which is needed to acquire the connection to the database. All of the RR classes implement the method doServe() and parse the URI, determining the type of resource the user wants to interact with.

The RR files also call the corresponding Data Access Objects (DAOs) that allows us to obtain different resources. We have one DAO for each RR file.

Concerning the Songs, in this diagram we have three REST classes that are: `AddSongsToAlbumsRR`, `AddSongsToPlaylistsRR` and `CreateSongsRR`, all of them extend the class `AbstractRR` that implements the interface `RestResource`. All of the RR files implements in their operations the corresponding DAO, in this case we have `AddSongsToAlbumsDAO`, `AddSongsToPlaylistsDAO` and `CreateSongsDAO`. All the DAO files extend the class `AbstractDAO` that implements the interface `DataAccessObject`. The rest of the classes have a similar structure. Overall, this design allows for separation of logic paths, with REST classes handling HTTP communication and DAOs handling database operations.

5.2 Sequence Diagram

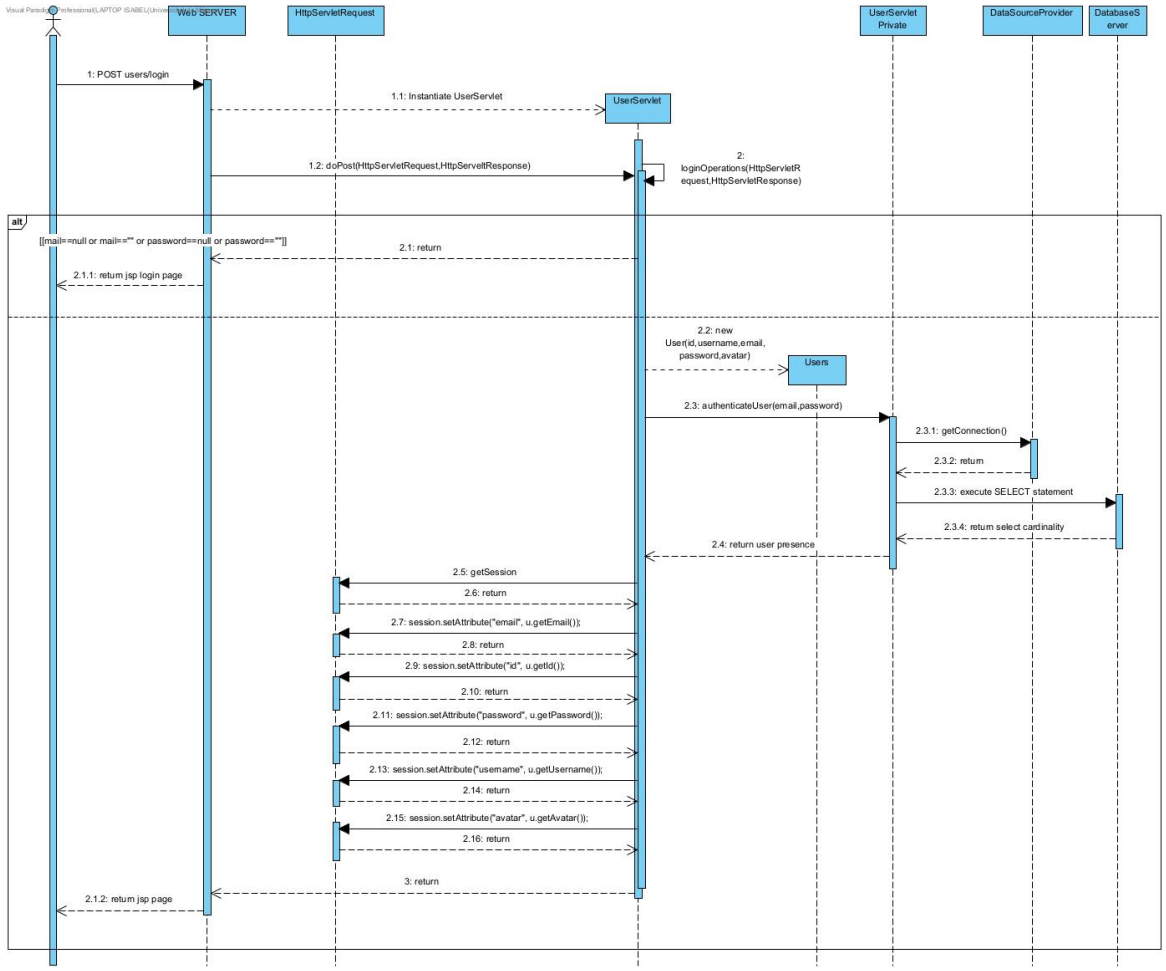

Figure 14: Class Diagram

Here reported the sequence diagram for the login. The user executes a POST request to the Web Server with the URI users/login. The data about the user that is trying to make the login (email and password) are passed to the web server. The web server initiates the UserServlet and calls the method doPost passing the attributes needed. The UserServlet analyzes the request and recognise it as a login operation, then it calls the method loginOperations. This method controls whether the email and password have been correctly provided, using the

private method authentificateUser (that has a connection with the DataSourceProvider) that access the database, if the data is correct, returns the login.jsp page, if not it shows a error. Else, a new user is initialized with the data. The method authentificateUser checks if in the database is a user with the email and password provided. After that, the control is returned to the UserServlet, which obtains the session for the HttpServletRequest and sets five parameters in the session, which are the attributes of the user.

5.3 REST API Summary

URI	Method	Description	Filter
- Albums -			
rest/albums	POST	Create album	-
rest/albums/artists/{ID}	GET	List albums by artist with specific ID	-
rest/albums/pickalgorithm	GET	List albums by algorithm	-
rest/albums/{ID}	DELETE	Delete album with specific ID	-
rest/albums/{ID}	GET	Info about album with specific ID	-
- Artists -			
rest/artists	POST	Create artist account	-
rest/artists/{ID}	DELETE	Delete artist account with specific ID	-
rest/artists/{ID}	GET	Info about artist with specific ID	-
rest/artists/{ID}	PUT	Update entry of artist with specific ID	-
- Playlists -			
rest/playlists	POST	Create playlist	-
rest/playlists/users/{ID}	GET	List playlists by user with specific ID	-
rest/playlists/{ID}	DELETE	Delete playlist with specific ID	-
rest/playlists/{ID}	GET	Info about playlist with specific ID	-
- Songs -			
rest/songs	POST	Create song	-
rest/songs/artists/{ID}	GET	List songs by artist with specific ID	-
rest/songs/pickalgorithm	GET	List songs by algorithm	-
rest/songs/albums/{ID}	GET	List songs in album with specific ID	-
rest/songs/albums/{ID}	POST	Add songs to album with specific ID	-
rest/songs/{ID}/playlists/{ID}	DELETE	Delete song with specific ID from playlist with specific ID	-
rest/songs/playlists/{ID}	GET	List songs in playlist with specific ID	-
rest/songs/playlists/{ID}	POST	Add songs to playlist with specific ID	-
rest/songs/{ID}	DELETE	Delete song with specific ID	-
rest/songs/{ID}	GET	Info about song with specific ID	-
- Users -			
rest/users	POST	Create user account	-
rest/users/{ID}/albums/{ID}	DELETE	User with specific ID un-likes album with specific ID	-
rest/users/{ID}/albums	GET	List albums liked by user with specific ID	-
rest/users/{ID}/albums	POST	Like album with specific ID	-
rest/users/{ID}/songs/{ID}	DELETE	User with specific ID un-likes song with specific ID	-
rest/users/{ID}/songs	GET	List songs liked by user with specific ID	-
rest/users/{ID}/songs	POST	Like song with specific ID	-
rest/users/{ID}	DELETE	Delete user account with specific ID	-
rest/users/{ID}	GET	Info about user with specific ID	-
rest/users/{ID}	PUT	Update entry of user with specific ID	-

Table 2: Description of REST API

!!! The final login and filter features are not yet implemented !!!

5.4 REST Error Codes

Error Code	HTTP Status Code	Description
-100	SC_BAD_REQUEST	Wrong format.
-101	SC_BAD_REQUEST	One or more input fields are empty.
-202	SC_BAD_REQUEST	Email missing.
-104	SC_BAD_REQUEST	Password missing.
-105	SC_BAD_REQUEST	Submitted credentials are wrong.
-107	SC_CONFLICT	Different passwords introduced.
-108	SC_CONFLICT	Email is already in use.
-209	SC_BAD_REQUEST	Username already exists.
-116	SC_BAD_REQUEST	Problems in creating playlist.
-117	SC_BAD_REQUEST	Problems in creating album.
-118	SC_BAD_REQUEST	Problems in creating artist.
-119	SC_BAD_REQUEST	Problems in creating song.
-120	SC_BAD_REQUEST	Problems in creating user.
-301	SC_BAD_REQUEST	Problems in updating user.
-302	SC_BAD_REQUEST	Problems in updating artist.
-303	SC_INTERNAL_SERVER_ERROR	Failed to upload lyrics.
-112	SC_BAD_REQUEST	Song not found.
-410	SC_INTERNAL_SERVER_ERROR	Failed to like the song.
-411	SC_BAD_REQUEST	Song already liked by the user.
-113	SC_BAD_REQUEST	Album not found.
-412	SC_INTERNAL_SERVER_ERROR	Failed to like the album.
-413	SC_BAD_REQUEST	Album already liked by the user.
-600	SC_BAD_REQUEST	Invalid Song ID.
-601	SC_BAD_REQUEST	Song title missing.
-602	SC_BAD_REQUEST	Invalid YouTube URL.
-603	SC_BAD_REQUEST	Invalid release date for the song.
-604	SC_BAD_REQUEST	Invalid duration for the song.
-700	SC_BAD_REQUEST	Invalid Artist ID.
-701	SC_BAD_REQUEST	Artist name missing.
-702	SC_BAD_REQUEST	Biography missing for the artist.
-703	SC_BAD_REQUEST	Verification status missing for the artist.
-704	SC_BAD_REQUEST	Language missing for the artist.
-800	SC_BAD_REQUEST	Invalid Album ID.
-801	SC_BAD_REQUEST	Album title missing.
-802	SC_BAD_REQUEST	Album image missing.
-803	SC_BAD_REQUEST	Invalid release date for the album.
-900	SC_BAD_REQUEST	Invalid Playlist ID.
-901	SC_BAD_REQUEST	Playlist title missing.
-1000	SC_BAD_REQUEST	Invalid User ID.
-1001	SC_BAD_REQUEST	Username missing.
-1004	SC_BAD_REQUEST	Avatar missing.
-1005	SC_BAD_REQUEST	Invalid avatar format.
-114	SC_BAD_REQUEST	Album not found.
-115	SC_BAD_REQUEST	Artist not found.
-121	SC_BAD_REQUEST	The input JSON is in the wrong format.

-400	SC_BAD_REQUEST	Invalid request.
-403	SC_FORBIDDEN	Access denied.
-401	SC_UNAUTHORIZED	Unauthorized access.
-500	SC_INTERNAL_SERVER_ERROR	Database error.
-999	SC_INTERNAL_SERVER_ERROR	Internal Error.

Table 3: Description of REST ERRORS

5.5 REST API Details

A resource

We report here three different resources types that our web application handles during its functioning.

Song

The following endpoint allows to insert a new song into the database

- URL: `songs/`
- Method: `POST`
- URL Parameters: no parameters are required in the url
- Data Parameters:
  Required:
  artistid = integer: The ID used for the artist that created the song. It should belong to artist already in the database.
- Success Response:
  Code: 200
  Content: data: songid : 16, artistid : 1
- Error Response:
  Code : 404 NOT FOUND
  Content : {"error": {"code": -115, "message" : "Artist not found ."}
  When: The user has passed an artist id which was not found in the database.

  Code : 500 INTERNAL SERVER ERROR
  Content : {"error": {"code": -999, "message" : "Internal Error."}
  When: if there is a SQLException or a NamingException.

List Albums by Artists

The following endpoint allows to get the list of albums that an artists has

- URL: `/rest/albums/artists/{ID}`
- Method: `GET`
- URL Parameters: ArtistID
- Data Parameters: None
- Success Response:
  Code: 200
  Content: data: [id : 1, title : "Shape of you",id : 35, title : "happier"...]

- Error Response:

  Code : 404 NOT FOUND
  Content : {“error”: {“code”: -115, “message” : ‘Artist not found .”}
  When: The URI has an artist id which was not found in the database.

  Code : 500 INTERNAL SERVER ERROR
  Content : {“error”: {“code”: -999, “message” : “Internal Error.”}
  When: if there is a SQLException or a NamingException.
  .

Delete album

The following endpoint allows to delete a album from the database

- URL: `/albums/{ID}`
- Method: `DELETE`
- URL Parameters:
  albumid = integer
- Data Parameters: None
- Success Response:
  Code: 200
  Content: The album is deleted from the database
- Error Response:
  Code : 404 NOT FOUND
  Content : {“error”: {“code”: -114, “message” : “Album not found .”}
  When: The URI has an album id which was not found in the database.

  Code : 500 INTERNAL SERVER ERROR
  Content : {“error”: {“code”: -999, “message” : “Internal Error.”}
  When: if there is a SQLException or a NamingException.
  .

6 Group Members Contribution

Isabel Torres-Pardo López sum up:

- LaTeX Homework file:
  - Objectives
  - Data Logic Layer : text + schemas (later revised by Álvaro Martín Bacas)
  - Business Logic Layer : descriptions + diagrams
- Helped in servlets files (together with Jesús Alcázar Pérez)
- Helped in jsp classes (together with Jesús Alcázar Pérez)
- Helped in doing the Sequence Diagram (together with Jesús Alcázar Pérez)
- database: `database/ranamelone.sql` (created initial tables later revised), `database/insert.sql`
- packages in `it.unipd.dei.ranamelone`:
  - `filter.*` (together with Jesús Alcázar Pérez)
  - `resources.*` (initial base later revised and perfected)
  - `rest.Create*`,
  - `rest.Delete*` (except `rest.DeleteSongFromPlaylistRR`)
  - `rest.ListAlbumsByArtistRR`, `rest.ListPlaylistsByUserRR`, `rest.ListSongsByArtistRR`
  - `rest.Update*`
  - `utils.DataSourceProvider`
  - `dao.*` (initial base later revised and perfected)

Jesús Alcázar Pérez sum up:

- LaTeX Homework file:
  - Objectives
  - Main Functionalities
  - Presentation Logic Layer: pages and logic definition + final mockups
- Helped in servlets files (together with Isabel Torres-Pardo López)
- Helped in jsp classes (together with Isabel Torres-Pardo López)
- Helped in doing the Sequence Diagram (together with Isabel Torres-Pardo López)
- packages in `it.unipd.dei.ranamelone`:
  - `resources.Actions`
  - `filter.*` (together with Isabel Torres-Pardo López)
  - `rest.Info*`

Andrea Valentinuzzi sum up:

- Repository structure setup (+ Abstract classes etc.)
- Presentation Logic Layer: pages and logic definition + initial approximate sketch
- REST API (together with Giovanni Brejc 50-50)
- LaTeX Homework file:
  - 5.3 REST API Summary
- files: `docker-compose.yml`, `pom.xml`, `web.xml`

- database: `database/ranamelone.sql` (updates and feature completions, together with Arjun Jassal and Giovanni Brejc)
- packages in `it.unipd.dei.ranamelone`:
  - `servlet.AbstractDatabaseServlet`, `servlet.RestDispatcherServlet`
  - `func.*` (together with Giovanni Brejc 50-50)
  - `resources.*` (except `resources.Actions`)
  - `rest.Like*`
  - `rest.ListAlbumsByAlgorithmRR`, `rest.ListAlbumsLikedByUserRR`, `rest.ListSongsByAlgorithmRR`, `rest.ListSongsLikedByUserRR`
  - `dao.Delete*` (except `dao.DeleteSongFromPlaylistDAO`)
  - `dao.List*`
- final notes (together with Arjun Jassal and Giovanni Brejc):
  - code refactor, syntax corrections, error codes checking over the whole project (total re-writing of `it.unipd.dei.ranamelone.*`)
  - bug fixing
  - some REST calls testing (just basic ones, TODO: complete testing)

Giovanni Brejc sum up:

- Presentation Logic Layer: pages and logic definition + initial approximate sketch
- REST API (together with Andrea Valentinuzzi 50-50)
- database: `database/ranamelone.sql` (updates and feature completions, together with Andrea Valentinuzzi and Arjun Jassal)
- packages in `it.unipd.dei.ranamelone`:
  - `func.*` (together with Andrea Valentinuzzi 50-50)
  - `rest.Unlike*`
  - `rest.ListSongsInAlbumRR`, `rest.ListSongsInPlaylistRR`
  - `dao.Like*`
  - `dao.Unlike*`
  - `dao.Update*`
- final notes (together with Andrea Valentinuzzi and Arjun Jassal):
  - code refactor, syntax corrections, error codes checking over the whole project (total re-writing of `it.unipd.dei.ranamelone.*`)
  - bug fixing

Arjun Jassal sum up:

- Presentation Logic Layer: pages and logic definition + initial approximate sketch
- database: `database/ranamelone.sql` (updates and feature completions, together with Andrea Valentinuzzi and Giovanni Brejc)
- packages in `it.unipd.dei.ranamelone`:
  - `rest.Add*`
  - `rest.DeleteSongFromPlaylistRR`
  - `dao.Add*`
  - `dao.DeleteSongFromPlaylistDAO`

  - `dao.Create*`
  - `dao.Info*`
- final notes (together with Andrea Valentinuzzi and Giovanni Brejc):
  - code refactor, syntax corrections, error codes checking over the whole project (total re-writing of `it.unipd.dei.ranamelone.*`)
  - bug fixing

Álvaro Martín Bacas sum up:

- UploadSong Mockup
- Updated final DB schema and tables and DBSchema latex file
- packages in `it.unipd.dei.ranamelone`:
  - `utils.ErrorCode`
  - `dao.*` (some files creation)
  - `rest.*` (some files creation)

TODO things not yet tackled/completed:

- finish testing all REST calls
- implement Login feature
- implement Filter feature
- sort content of `it.unipd.dei.ranamelone`'s sub-packages in sub-sub-packages for better readability

NOTE(s) Disclaimers:

- (most of the) Contributions from Andrea Valentinuzzi, Arjun Jassal and Giovanni Brejc were submitted to BitBucket with the 'akappakappa' account, that belongs to Andrea Valentinuzzi, but were worked on together by the 3 students using the "Code With Me" feature inside IntelliJ IDEA.
  Those commits' messages contain: "Code With Me: Andrea Valentinuzzi andrea.valentinuzzi@studenti.unipd.it, Giovanni Brejc giovanni.brejc@studenti.unipd.it, Arjun Jassal arjun.jassal@studenti.unipd.it".